Robot Programming with Lisp

1. Introduction, Setup

Arthur Niedzwiecki

Institute for Artificial Intelligence
University of Bremen

20th October, 2020

General Info

- Lecturer: Arthur (PhD student at IAI)
- Correspondence: `aniedz@cs.uni-bremen.de`
- Dates: Thursdays, 14:15 - 15:45, 16:15 - 17:45
- Language: English and German
- Credits: 6 ECTS (4 SWS)
- Course type: practical course
- Course number: 03-IBVP-RPWL (03-BE-710.98b)
- Location: TAB Building, Room 0.30 EG

Plan

Introduction

Course Content

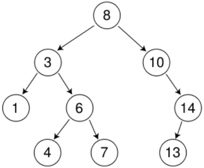
Organizational

Assignment

Course content

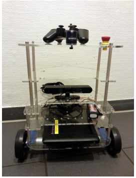
Common Lisp

Artificial Intelligence



Robot Operating System (ROS)

Robot platform



Common Lisp

- Full-featured industry-standard programming language

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing
- Compiles into machine code

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing
- Compiles into machine code
- Best choice for symbolic processing (AI, theorem proving, etc.)

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing
- Compiles into machine code
- Best choice for symbolic processing (AI, theorem proving, etc.)
- Good choice for writing domain-specific programming languages (e.g., robot programming languages)

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing
- Compiles into machine code
- Best choice for symbolic processing (AI, theorem proving, etc.)
- Good choice for writing domain-specific programming languages (e.g., robot programming languages)

Applications using / written in dialects of Lisp:
Emacs, AutoCAD, Grammarly, Mirai (Gollum animation), Google ITA (airplane ticket price planner AI), DART (DARPA logistics AI), Maxima (computer algebra system), AI frameworks, NASA satellites ...

ROS

- Middleware for communication of the components of a robotic system

ROS

- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)

ROS

- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)
- Powerful build system (based on CMake), with a strong focus on integration and documentation

ROS

- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)
- Powerful build system (based on CMake), with a strong focus on integration and documentation
- Language-independent architecture: C++, Python, Lisp and more

ROS

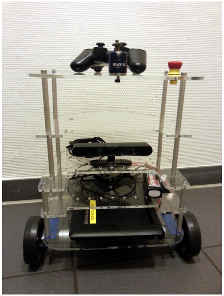
- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)
- Powerful build system (based on CMake), with a strong focus on integration and documentation
- Language-independent architecture: C++, Python, Lisp and more
- According to ROS 2020 Community Metrics Report,
  - More than 2 million unique pageviews `wiki.ros.org` a month
  - More than 38 million downloads of `.deb` packages a month

ROS

- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)
- Powerful build system (based on CMake), with a strong focus on integration and documentation
- Language-independent architecture: C++, Python, Lisp and more
- According to ROS 2020 Community Metrics Report,
  - More than 2 million unique pageviews `wiki.ros.org` a month
  - More than 38 million downloads of `.deb` packages a month
- *De facto* standard in modern robotics

TortugaBot

- 2 controllable wheels
- 2D laser scanner
- Thinkpad E485 PC with bluetooth
- PlayStation joystick



Why Lisp for robots?

- ROS supports a number of languages

Why Lisp for robots?

- ROS supports a number of languages
- Lisp is good for rapid prototyping

Why Lisp for robots?

- ROS supports a number of languages
- Lisp is good for rapid prototyping
- It is more suitable for symbolic reasoning and AI

Why Lisp for robots?

- ROS supports a number of languages
- Lisp is good for rapid prototyping
- It is more suitable for symbolic reasoning and AI
- There are existing robot programming languages in Lisp that automate decision making

Rough schedule

Assignments (single, this year)

- Introduction & Setup
- Lisp basics
- OOP & Failure Handling
- Functional programming
- Search Algorithms

Rough schedule

Assignments (single, this year)

- Introduction & Setup
- Lisp basics
- OOP & Failure Handling
- Functional programming
- Search Algorithms

Intermediate (until mid Jan '22)

- ROS Lisp API (*roslisp*)
- 2D world of *turtlesim*
- Coordinate frames of *TF*

Rough schedule

Assignments (single, this year)

- Introduction & Setup
- Lisp basics
- OOP & Failure Handling
- Functional programming
- Search Algorithms

Intermediate (until mid Jan '22)

- ROS Lisp API (*roslisp*)
- 2D world of *turtlesim*
- Coordinate frames of *TF*

Project (groups, Jan-Feb '22)

- Controlling TortugaBot
- Reading sensor data
- Collision avoidance
- Heuristic decision-making
- The big day: competition

Course Goals

You will learn / improve your skills in the following:

- Common Lisp, of course
- Git
- Functional programming
- Cognitive robotics
- Jupyter Notebook
- Docker
- Linux
- ROS (for future roboticists)
- Emacs (the IDE for Lisp devs)

...and get to play with a real little robot!

Plan

Introduction

Course Content

Organizational

Assignment

Grading

- Course final grade: 100 points = 50 homework + 50 group project.

Grading

- Course final grade: 100 points = 50 homework + 50 group project.
- To participate in the project you need at least 25 points from the homeworks, otherwise it's a fail.

Grading

- Course final grade: 100 points = 50 homework + 50 group project.
- To participate in the project you need at least 25 points from the homeworks, otherwise it's a fail.
- Final grade: 50 of 100 points - 4.0, 100 of 100 points - 1.0.

Grading

- Course final grade: 100 points = 50 homework + 50 group project.
- To participate in the project you need at least 25 points from the homeworks, otherwise it's a fail.
- Final grade: 50 of 100 points - 4.0, 100 of 100 points - 1.0.
- $Grade = \dfrac{(100 - P_{your})}{(100 - 50)} * 3 + 1$

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitHub.

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitHub.
- The code you write should be uploaded to GitHub (`https://github.com/`).

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitHub.
- The code you write should be uploaded to GitHub (`https://github.com/`).
- Homework is due in one week.

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitHub.
- The code you write should be uploaded to GitHub (`https://github.com/`).
- Homework is due in one week.
- Solutions are discussed in the tutorial.

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitHub.
- The code you write should be uploaded to GitHub (`https://github.com/`).
- Homework is due in one week.
- Solutions are discussed in the tutorial.
- Can get 60 of 50 points in homework (can skip one homework).

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitHub.
- The code you write should be uploaded to GitHub (`https://github.com/`).
- Homework is due in one week.
- Solutions are discussed in the tutorial.
- Can get 60 of 50 points in homework (can skip one homework).
- Bonus points for very good homework solutions.

Scheinbedingungen Summary

- Graded homework every week until January, then group project
- Live presentation of the group project, individual grading
- 50 homework + 50 group project = 100 points for final grade
- homeworks have 60 points total, so there's a buffer if you miss one
- at least 25 points from the homeworks
- Final grade: 50 of 100 points - 4.0, 100 of 100 points - 1.0.
- $Grade = \frac{(100 - P_{your})}{(100 - 50)} * 3 + 1$

Links

- This lectures website:
  https://ai.uni-bremen.de/teaching/cs-lisp-ws22
- Git reference book:
  https://git-scm.com/docs/gittutorial
- Lisp books:
  http://landoflisp.com/, http://www.paulgraham.com/onlisp.html, http://www.gigamonkeys.com/book/
- Emacs cheat sheet:
  https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

Info summary

Next class:

- Date: 27.10. **???**
- Time: 14:15 (14:00 - 14:15 for questions)
- Place: same room (TAB 0.30)

Assignment:

- Due: 26.10, Wednesday, 23:59 **???**
- Points: 3 points
- For questions: write me a mail
  or ask your colleagues in the StudIP forum

Plan

Introduction

Course Content

Organizational

Assignment

Assignment goals

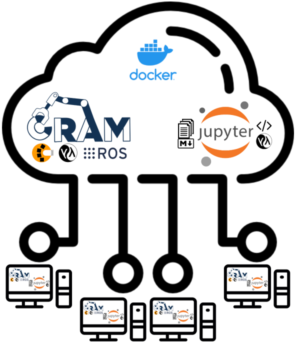
Set up your working environment

Set up your Git repository

Get comfortable with Jupyter

Cognitive Robotics for everyone

Docker is a manager vor virtual machines.
DockerHub hosts the virtual machine, ready to be downloaded



Task 1: Get Docker

Task 1: Get Docker

Depending on your system you can get Docker in different ways.
Follow `https://github.com/cram2/cram_teaching#readme` for details

- Linux (Debian 10-12, Ubuntu 18.04-22.04)
  Install docker-compose via CLI

Task 1: Get Docker

Depending on your system you can get Docker in different ways.
Follow `https://github.com/cram2/cram_teaching#readme` for details

- Linux (Debian 10-12, Ubuntu 18.04-22.04)
  Install docker-compose via CLI
- Windows 11
  Install docker-compose via PowerShell

Task 1: Get Docker

Depending on your system you can get Docker in different ways.
Follow `https://github.com/cram2/cram_teaching#readme` for details

- Linux (Debian 10-12, Ubuntu 18.04-22.04)
  Install docker-compose via CLI
- Windows 11
  Install docker-compose via PowerShell
- Windows 10
  Use WSL to get Ubuntu, then install Docker
  Or try installing docker-compose via PowerShell too

Task 1: Get Docker

Depending on your system you can get Docker in different ways.
Follow `https://github.com/cram2/cram_teaching#readme` for details

- Linux (Debian 10-12, Ubuntu 18.04-22.04)
  Install docker-compose via CLI
- Windows 11
  Install docker-compose via PowerShell
- Windows 10
  Use WSL to get Ubuntu, then install Docker
  Or try installing docker-compose via PowerShell too
- MacOS
  If you have an ARM M1 CPU check out these notes here:
  `https://docs.docker.com/desktop/mac/apple-silicon/`

Task 1 Check: Test if Docker works

- On Linux and older installations:

  ```
  docker-compose version
  ```

- On newer and other (e.g. Windows, Rosetta):

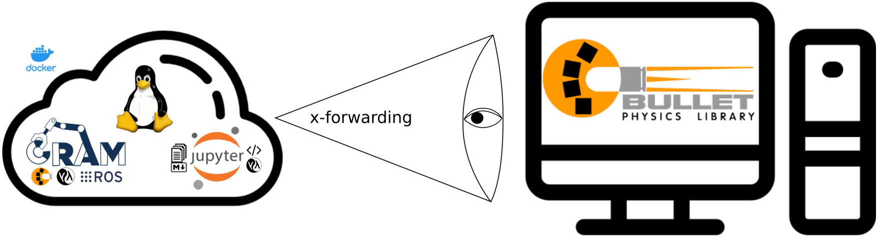
  ```
  docker compose version
  ```

- Check rights

  ```
  docker run hello-world
  ```

Task 2: Configure X-Forwarding

Visual applications run in the virtual machine (Docker container) using X, which is a visualization technique for Linux systems. Docker can't visualize itself, so we forward the Bullet Physics Simulation to your PC.



Task 2: Configure X-Forwarding

Follow `https://github.com/cram2/cram_teaching#readme` for details

- Linux (Debian 10-12, Ubuntu 18.04-22.04)

```
sudo apt install x11-xserver-utils
xhost +local:docker
```

Task 2: Configure X-Forwarding

Follow `https://github.com/cram2/cram_teaching#readme` for details

- Linux (Debian 10-12, Ubuntu 18.04-22.04)

  ```
  sudo apt install x11-xserver-utils
  xhost +local:docker
  ```

- Windows

  Install and configure VcXsrv, add Firewall rule

Task 2: Configure X-Forwarding

Follow `https://github.com/cram2/cram_teaching#readme` for details

- Linux (Debian 10-12, Ubuntu 18.04-22.04)

```
sudo apt install x11-xserver-utils
xhost +local:docker
```

- Windows
  Install and configure VcXsrv, add Firewall rule
- MacOS
  ?

Task 3: Git

Git provides version-control of changing code. A Git repository is a storage place for code. With Git it is easy to manage group projects and keep track of changes.

`https://git-scm.com/book/en/v2/Getting-Started-Installing-Git`

Using Git via CLI provides the best experience to understand how it works. There are also Git clients with a GUI. This lecture will only cover the CLI commands for Git.

Task 3: Git and GitHub Setup

- Create an account on GitHub if you don't have one:
  `https://github.com/`

Task 3: Git and GitHub Setup

- Create an account on GitHub if you don't have one: `https://github.com/`
- Create a new repository, call it `lisp_course_exercises`. Make it private.

Task 3: Git and GitHub Setup

- Create an account on GitHub if you don't have one:
  `https://github.com/`
- Create a new repository, call it `lisp_course_exercises`.
  Make it private.
- In project "Settings" → "Collaborators" add
  "Arthur Niedzwiecki (artnie)" as collaborator.

Task 3: Git and GitHub Setup

- Create an account on GitHub if you don't have one:
  `https://github.com/`
- Create a new repository, call it `lisp_course_exercises`. Make it private.
- In project "Settings" → "Collaborators" add "Arthur Niedzwiecki (artnie)" as collaborator.
- Install Git:
  `https://git-scm.com/book/en/v2/Getting-Started-Installing-Git`

Task 4: Git and Lecture Content

- On your PC, choose where to put the lectures project.

  ```
  cd into/the/desired/directory
  ```

Task 4: Git and Lecture Content

- On your PC, choose where to put the lectures project.
  ```
  cd into/the/desired/directory
  ```
- Download the course material:
  ```
  git clone https://github.com/cram2/cram_teaching.git lisp_course_exercises
  ```

Task 4: Git and Lecture Content

- On your PC, choose where to put the lectures project.

```
cd into/the/desired/directory
```

- Download the course material:

```
git clone https://github.com/cram2/cram_teaching.git lisp_course_exercises
```

- Define a remote target with the address of your new GitHub repo:

```
cd lisp_course_exercises
```

Replace YOUR_GITHUB_USERNAME in the following command.

```
git remote add my https://github.com/YOUR_GITHUB_USERNAME/lisp_course_exercises.git
```

Task 4: Git and Lecture Content

- On your PC, choose where to put the lectures project.

  ```
  cd into/the/desired/directory
  ```

- Download the course material:

  ```
  git clone https://github.com/cram2/cram_teaching.git lisp_course_exercises
  ```

- Define a remote target with the address of your new GitHub repo:

  ```
  cd lisp_course_exercises
  ```

  Replace YOUR_GITHUB_USERNAME in the following command.

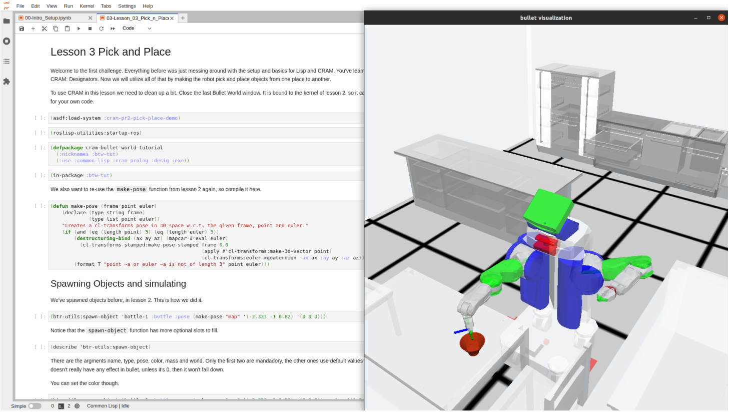
  ```
  git remote add my https://github.com/YOUR_GITHUB_USERNAME/lisp_course_exercises.git
  ```

- Upload the files to your new GitHub repo:

  ```
  git push -u my main
  ```

Task 5: Get into Jupyter & Test your setup

Jupyter combines code with documentation. Each lesson is a mix of Markdown plain text, and executable Lisp code.



Lesson 3 Pick and Place

Welcome to the first challenge. Everything before was just messing around with the setup and basics for Lisp and CRAM. You've learn
CRAM: Designators. Now we will utilize all of that by making the robot pick and place objects from one place to another.

To use CRAM in this lesson we need to clean up a bit. Close the last Bullet World window. It is bound to the kernel of lesson 2, so it ca
for your own code.

```
[ ]: (asdf:load-system :cram-pr2-pick-place-demo)
```

```
[ ]: (roslisp-utilities:startup-ros)
```

```
[ ]: (defpackage cram-bullet-world-tutorial
       (:nicknames :btw-tut)
       (:use :common-lisp :cram-prolog :desig :exe))
```

```
[ ]: (in-package :btw-tut)
```

We also want to re-use the `make-pose` function from lesson 2 again, so compile it here.

```
[ ]: (defun make-pose (frame point euler)
         (declare (type string frame)
                  (type list point euler))
         "Creates a cl-transforms pose in 3D space w.r.t. the given frame, point and euler."
         (if (and (eq (length point) 3) (eq (length euler) 3))
             (destructuring-bind (ax ay az) (mapcar #'eval euler)
               (cl-transforms-stamped:make-pose-stamped frame 0.0
                                                        (apply #'cl-transforms:make-3d-vector point)
                                                        (cl-transforms:euler->quaternion :ax ax :ay ay :az az))
             (format T "point ~a or euler ~a is not of length 3" point euler)))
```

Spawning Objects and simulating

We've spawned objects before, in lesson 2. This is how we did it.

```
[ ]: (btr-utils:spawn-object 'bottle-1 :bottle :pose (make-pose "map" '(-2.323 -1 0.82) '(0 0 0)))
```

Notice that the `spawn-object` function has more optional slots to fill.

```
[ ]: (describe 'btr-utils:spawn-object)
```

There are the argments name, type, pose, color, mass and world. Only the first two are mandadory, the other ones use default values
doesn't really have any effect in bullet, unless it's 0, then it won't fall down.

You can set the color though.

Task 5: Get into Jupyter & Test your setup

- With a terminal in the repository, check if your files look like the repository on github.
  Linux & Mac: `ls -la`
  Windows: `dir`

Task 5: Get into Jupyter & Test your setup

- With a terminal in the repository, check if your files look like the repository on github.
  Linux & Mac: `ls -la`
  Windows: `dir`
- Start docker-compose where the "docker-compose.yml" is.
  Linux: `docker-compose up`
  Win & Mac: `docker compose up`
  This will download the virtual machine and boot it. When done, enter the URL at the end into your browser. This is Jupyter Notebook.

Task 5: Get into Jupyter & Test your setup

- With a terminal in the repository, check if your files look like the repository on github.
  Linux & Mac: `ls -la`
  Windows: `dir`
- Start docker-compose where the "docker-compose.yml" is.
  Linux: `docker-compose up`
  Win & Mac: `docker compose up`
  This will download the virtual machine and boot it. When done, enter the URL at the end into your browser. This is Jupyter Notebook.
- Start the X-Forwarding
  Linux: `xhost +local:docker`
  Windows: Configure and start VcXsrv and allow via Firewall seetings.

Task 5: Get into Jupyter & Test your setup

- With a terminal in the repository, check if your files look like the repository on github.
  Linux & Mac: `ls -la`
  Windows: `dir`
- Start docker-compose where the "docker-compose.yml" is.
  Linux: `docker-compose up`
  Win & Mac: `docker compose up`
  This will download the virtual machine and boot it. When done, enter the URL at the end into your browser. This is Jupyter Notebook.
- Start the X-Forwarding
  Linux: `xhost +local:docker`
  Windows: Configure and start VcXsrv and allow via Firewall seetings.
- In Jupyter, navigate to "lectures/tutorials/00-Intro_Setup.ipynb"
  Go through the setup guide. If the demo at the end runs, your good!

Task 6: Get familiar with Git

- Go to `lectures/robot_programming_with_lisp/01_orc_battle/` and play it.

Task 6: Get familiar with Git

- Go to `lectures/robot_programming_with_lisp/01_orc_battle/` and play it.
- Check what's new in your local repo with `git status`

Task 6: Get familiar with Git

- Go to `lectures/robot_programming_with_lisp/01_orc_battle/` and play it.
- Check what's new in your local repo with `git status`
- Check detailed filechanges with `git diff` (q to exit):

Task 6: Get familiar with Git

- Go to `lectures/robot_programming_with_lisp/01_orc_battle/` and play it.
- Check what's new in your local repo with `git status`
- Check detailed filechanges with `git diff` (*q* to exit):
- The red files are the new untracked ones, the green ones are already in the Git index. To add new files to the index use `git add .`

Task 6: Get familiar with Git

- Go to `lectures/robot_programming_with_lisp/01_orc_battle/` and play it.
- Check what's new in your local repo with `git status`
- Check detailed filechanges with `git diff` (*q* to exit):
- The red files are the new untracked ones, the green ones are already in the Git index. To add new files to the index use `git add .`
- If you deleted some files, to remove them with `git add -u`

Task 6: Get familiar with Git

- Go to `lectures/robot_programming_with_lisp/01_orc_battle/` and play it.
- Check what's new in your local repo with `git status`
- Check detailed filechanges with `git diff` (*q* to exit):
- The red files are the new untracked ones, the green ones are already in the Git index. To add new files to the index use `git add .`
- If you deleted some files, to remove them with `git add -u`
- Once you're sure the changes are final, commit **locally**:

```
git commit -m "A meaningful commit message."
```

Task 6: Get familiar with Git

- Go to `lectures/robot_programming_with_lisp/01_orc_battle/` and play it.
- Check what's new in your local repo with `git status`
- Check detailed filechanges with `git diff` (*q* to exit):
- The red files are the new untracked ones, the green ones are already in the Git index. To add new files to the index use `git add .`
- If you deleted some files, to remove them with `git add -u`
- Once you're sure the changes are final, commit **locally**:
  ```
  git commit -m "A meaningful commit message."
  ```
- Finally, to **upload** your local commits to the Github server, push the changes upstream:
  ```
  git push
  ```

Troubleshoot

For troubleshooting, consider the setup documention here:

`https://github.com/cram2/cram_teaching#readme`

or use the forum to work with your colleagues or write me a mail.

Q & A

Thanks for your attention!